THE MEDICAL QUERY LANGUAGE

Daniel J. Shusman, Mary M. Morgan
Rita Zielstorff, R.N., M.S., G. Octo Barnett, M.D.

Laboratory of Computer Science
Massachusetts General Hospital
Boston, Massachusetts 02114

Abstract

The Medical Query Language (MQL) is an English-like query language with which a user with little or no training in programming or computer science can formulate and satisfy inquiries on data contained in his/her Standard MUMPS database. To date, major applications of MQL have been in the areas of quality assurance, medical research, and practice administration at sites using the COmputer STored Ambulatory Record (COSTAR) database system.

Introduction

Because of increasing reliance on database management systems, it has become important that ad hoc access to the information be available. Some reporting packages delivered with systems are designed to satisfy specific needs (e.g., the COSTAR Encounter and Status Reports). Others are designed to accept certain user specification of content and format (e.g., the COSTAR Report Generator). The last alternative is the submittal of report specifications to a data processing staff, if one is available. None of these approaches provide the user with the necessary flexibility to query a database in an ad hoc fashion to the depth of detail desired.

MQL was developed in response to this situation. Users are able to satisfy their information needs in a very reasonable time without learning programming, without resorting to "packaged" report programs, and without diverting the data processing staff from its primary work.

MQL provides full data evaluation capability and complex branching logic without sacrificing ease of use. Queries may be indefinitely long and intricate. Such queries can be broken down into a series of subqueries, each designed to accomplish some portion of the total problem.

Design

Much research has been carried out in the development of query languages. Non-medical systems requiring a formal syntactical and/or procedural approach to queries (for example, QUEL[1], SEQUEL[2], and OBE[3]) and "natural language" systems (for example, RENDEZVOUS[4], LIFER[5], and INTELLECT[6]) have been developed.

There are systems designed specifically to facilitate inquiry of dedicated medical information databases. Such systems include CLINFO[7], MEDINFO[8], MISAR[9], and WISAR[10].

Other, more general query systems are provided by MEDUS/A[11] (a hierarchical database management system whose query language is quite symbolic) and MEDQUEL[12] (a "natural language" interface to the QUEL query language).

MQL was designed as a high-level, English-like query language. It does not require of the user an in-depth knowledge of the subject database structure, but imposes a well-defined source statement syntax and requires a procedural expression of the question.

MQL uses a source vocabulary supplied by two dictionaries; the first being a command and data operator lexicon, the second being the schema - a description of the data fields contained in the subject database. Each of these dictionaries provides descriptive, English-like names for the items contained in it. Synonyms for the primary names of these items can be incorporated, thus tailoring MQL's vocabulary to the terminology of its users.

The procedural nature of MQL imposes certain logical constraints on the user. In "natural language" systems ambiguities in the expression of a problem (the user's conceptualization) and the processor's interpretation of the query cannot always be easily resolved. Arguably, the procedural characteristic of MQL aids in developing the protocol and serves to add a structure to the query which later helps in identifying and correcting mistakes or in modifying the query's logic.

The User's Manual and other documentation provide the user with extensive information about, and many examples of, the procedural rules. MQL is therefore available to persons with little or no computing experience after only a short familiarization period. The language, however, is a tool powerful enough to warrant use by experienced programmers who will find a significant decrease in time expended developing

0195-4210/83/0000/0742$01.00 © 1983 IEEE

certain applications.

Implementation

The MQL language processor is comprised of a lexical scanner, a parser which results in an internal, Reverse Polish form of the source statement, and a 4-segment compiler which generates the appropriate, executable MUMPS routines using a macro-expansion technique.

As with many query languages, MQL has been implemented to maintain independence of the language processor and the subject database structure. This is accomplished by modeling the database in a file or "schema" (mentioned above). Each data field (or "attribute") in the database has an entry in the schema, as do all functions such as LENGTH(of text) and NUMERIC VALUE(of item).

The schema entry for an attribute includes one internal name, one or more external names, the data type (date, text, numeric, coded), a print format, and at least one definition. A definition represents one relationship of the attribute to other attributes in the database. Two basic items comprise a definition. The first is a "dependency list". A dependency list is a list of attributes required in the query before the definition can be selected (as below). The second item in a definition is the MUMPS code (macro) which will extract data from the database for the attribute. For example, if attribute A is "pointed to" by attributes B and C, then the schema entry for A will contain two definitions. The first will have a dependency list of B and the MUMPS macro describing access to A given B. The second will have a dependency list of C and a MUMPS macro describing A given C.

By having all relationships among all attributes completely defined, a user can be released from an in-depth knowledge of the structure of the underlying database. When composing a query, a user can assume a particular relationship between attributes. The language processor scans each definition of the attribute to which it is currently pointing in an attempt to select that one whose dependency list most nearly matches all attributes which the processor has already encountered (resolved) in the query. If the dependency list exactly matches the list of resolved attributes, the selection process terminates. If the definition is chosen because it most closely, but not exactly, matches the list of resolved attributes, those attributes required by the definition, but not already resolved, will be placed on the "resolved" list. The selection process then terminates for the current attribute. This is selection-by-inference and permits the user to assume attribute relationships. If no definition can be chosen, an error message is displayed and the user must intervene. In such a case the Attribute Tracking option is available to point out all existing definitions for the attribute.

As mentioned, MQL is implemented as a translator which compiles query statements togenerate one or more executable "object" routines. The translation process is basically two-step:

1) Each statement is parsed for syntax and lexical validity. Attributes are verified. Coded values are translated to their internal forms. Any syntax errors or unrecognized terms are identified and the offending statement is presented to the user for correction.

2) The parsed version is then compiled into a series of MUMPS routines during which:

   a) MUMPS variables are substituted for attributes and variable names. (Local variables in MQL are bracketed -e.g., [X].)

   b) The internal branching logic required by the "top-down" execution path and by the use of subqueries (synonymous to traditional subroutines) is established, and

   c) The MUMPS macro code from the appropriate dependencies of all attributes required by the query (either explicitly named by the user in a statement or inferred by the selection process above) are concatenated in the correct logical order and are filed in the object routine(s).

Once compiled, a query can be executed by issuing the RUN command which allows immediate or delayed execution.

User Interaction

MQL is delivered with a query editor which has display, edit, and erase commands. Using the editor the query writer enters a source statement in the form:

<statement number><space><command keyword><command argument> [<;><command keyword><command argument>...]

where:

<statement number> is a source line sequencing number important to editing and establishing query logic.

<command keyword> is one of a group of valid MQL commands of which the following is a partial list:

FOR EACH:	introduces an iterative expression
WHEN:	introduces a predicate expression
DEFINE:	defines a subquery
DO:	causes execution of a subquery
LIST:	generates an output listing
STORE:	collects data in intermediate files
RETRIEVE:	recovers STOREd data.

<command argument> is an expression of a syntax appropriate to the preceding <command keyword>.

<;> is an optional delimiter allowing the placement of more than one logical source statement on one physical line.

Simple Examples

MQL has been field-tested at many sites using the COmputer STored Ambulatory Record (COSTAR)[13] database. The following examples are based on the schema which describes the structure of that database.

Example I. Identify all patients of Dr. Smith who have been diagnosed in the last year as having tuberculosis.

```
10  FOR EACH PATIENT ; WHEN PRIMARY MD IS SMITH
20  WHEN DATE AFTER TODAY-1 YEAR, DIVISION IS DX,
      CODE IS TUBERCULOSIS
30  LIST NAME, UNIT NUMBER, DATE, STATUS
```

The most interesting feature of this query is the resolution of attributes (i.e., choosing the appropriate definition of each attribute named by the user). By the selection-by-inference process described above, the writer of this query need not know that the attribute ENCOUNTER was required to make available DATE nor that the attribute EVENT need be included to recover STATUS. The user need only realize that only patients of Dr. Smith will be considered, that only diagnoses in the last year will be searched, and that only a diagnosis of tuberculosis in the last year will qualify the patient for inclusion in the report. This continual refinement of the data search - each attribute being subject to the values of all attributes preceding it in the query - is termed the CONTEXT under which each attribute is assigned its value(s).

Important to medical research is the comparison of data over time. MQL has a syntax named SUBCONTEXT which is expressly designed to accommodate such investigation.

Example II. Identify all patients of Dr. Smith who have been diagnosed in the last year as hypertensive and list all medications prescribed after the diagnosis.

```
10  FOR EACH PATIENT ; WHEN DIVISION1 IS DIAGNOSIS
20  WHEN CODE1 IS HYPERTENSION, DATE1 AFTER TODAY
      - 1 YEAR
30  WHEN STATUS1 NOT CONTAINS ERROR
40  LIST NAME, UNIT NUMBER, DATE1
50  DO MEDICATIONS ; NEXT PATIENT
60  /
70  DEFINE MEDICATIONS
80  WHEN DIVISION2 IS MEDICATIONS , DATE2 AFTER
      DATE1
90  LIST CODE2
```

The suffix "1" attached to DIVISION, CODE, DATE, AND STATUS "group" these attributes in the search for a diagnosis of hypertension. The suffix "2" attached later to DIVISION, CODE, AND DATE serves to allow the search for medications without contending with the values given to DIVISION, CODE, and DATE during the hypertension search. Thus, all EVENTS (selected by inference) in any one patient record can be searched independently; first for hypertension (subcontext 1) and then for medications (subcontext 2). When a medication EVENT is found, and its date (DATE2) is after the date of diagnosis (DATE1), the medication is listed, and the next medication EVENT is sought.

Of interest is line 50 above. The search for medications is done in a subquery to allow listing of all medication codes before returning to the main body of the query. When all medications meeting the date constraint in line 80 have been listed, the subquery terminates. The query then continues with the statement following the semicolon in line 50. NEXT PATIENT is used to end the search for hypertension diagnoses in the current patient record by forcing the query to move on to the next patient record.

Hardware Environment

During the field testing, MQL (Version 1) has been run under DSM (Versions 1 and 2) and ISM and on PDP 11, VAX 11/780 and PRIME computers. MQL should run equally successfully under any Standard MUMPS system with modifications required to accommodate such items as non-standard "Z" commands.

MQL (Version 2), as delivered with the schema describing the current public domain COSTAR, requires a minimum partition size of 5K bytes, a minimum 512K bytes of disk space (for the lexicon and schema), and at least a 200K byte transient disk space. The size of schemata for other Standard MUMPS databases will change the disk requirements.

Recent Experience

Our experience over the time that MQL (Version 1) has been field-tested has been encouraging. Users report a high success rate in formulating and running both medical and administrative queries. Requests ranging from calling the COSTAR Status Report from a query to a scatter plot facility have been received and honored.

Under development is a prototypical, frame-driven user interface which will optionally replace the current source statement input mode. This interface represents each entity and associated attributes in the database as a distinct frame on which the user can enter value restrictions. The system releases the user from the concerns of procedure, context, and subcontext. However, upon completing the specifications, the user is presented with the source statement text equivalent of the frame specifications. The goal is to train such a user

in the use of the source statement input mode because it affords a much higher degree of flexibility.

Summary

The Medical Query Language is a comprehensive programming system which provides great flexibility in the retrieval, analysis, and reporting of information maintained in a Standard MUMPS database. Replete with full user and programmer documentation and such capabilities as cross-tabulation reports, scatter plots, on-line help, intermediate data storage, and system maintenance utilities, MQL can satisfy virtually all reporting needs of its users.

Acknowledgements

This project was supported in part by the National Center for Health Services Research under Grant 5 R18 HS 04073.

References

1. Held GD, Stonebraker MR, Wong E. INGRES - A Relational Data Base System. AFIPS - 1975, National Computer Conferences Proc., AFIPS Press, New Jersey, May, 1975.

2. Boyce R, Chamberlin D. SEQUEL - A Structured English Query Language. Proc. of the 1974 ACM-SIGEIDET Workshop on Data Description, Access, and Control, Ann Arbor, Michigan, May, 1975.

3. Zloof M. Office-by-Example: A business language that unifies data and word processing and electronic mail. IBM Systems Journal, Volume 21, Number 3, 1982 pp: 272-304.

4. Codd EF. Seven Steps to RENDEZVOUS with the Casual User. IBM Research Report RJ 1333, IBM San Jose Research Laboratory, 1974.

5. Hendrix GG, Sacerdoti ED, Sagalowicz D, Slocum J. Developing a Natural Language Interface to Complex Data. ACM Transactions on Database Systems, 1978, 3:106-147.

6. Artificial Intelligence Business Week Magazine, March 8, 1982, pg: 60.

7. Palley NA, Grover GF, Sibley WL, Hopwood MD. CLINFO Users Guide: Release One. R-1543-1-NIH, Rand Corp., April, 1976.

8. Johnson DC, Barnett GO. MEDINFO - A MEDical INFormation System. Laboratory of Computer Science, Massachusetts General Hospital, Boston, 1976.

9. Karpinski RHS, Bleich, HL. MISAR: A Miniature Information Storage and Retrieval System. Comput Biomed Res, 1971, 4:655-660.

10. Entine SM. Wisconsin Storage and Retrieval System, A Data Management System for a Clinical Cancer Center. Proceedings of the Sixth Annual Symposium of Computer Applications in Medical Care, Washington, D.C., 1982 pp: 813-814.

11. Goldstein L, Miller PB, Strong RM. MEDUS/A The MEDIQ Query Language. Technical Report 2, Version 2.4, Health Systems Project, Harvard School of Public Health, Boston, May, 1979.

12. Epstein MN, Walker DE. Natural Language Access to a Melanoma Data Base. Proceedings of the Second Annnual Symposium on Computer Applications in Medical Care. 1978, pp: 320-325.

13. Beaman PD, Justice NS, Barnett GO. A Medical Information System and Data Language for Ambulatory Practices. COMPUTER, 1979, 12:9-17.